Developer Technical Support

# Apple IIGS
# #71: DA Tips and Techniques

Revised by: Dave “Mr. Tangent” Lyons May 1992
Written by: Dave Lyons November 1989

This Technical Note presents tips and techniques for writing Desk Accessories.
**Changes since December 1991:** Reworked discussion of NDAs and Command- keystrokes. Marked obsolete steps in “NDAs Can Have Resource Forks.”

## Classic Desk Accessory Tips and Techniques

### Reading the Keyboard

For a CDA that runs only under GS/OS, the Console Driver is the best choice for reading from the keyboard. Other CDAs have two cases to deal with: the Event Manager may or may not be started. The Text Tools can read the keyboard in either case, but you should avoid using the Text Tools whenever possible (see Apple IIGS Technical Note #69, The Ins and Outs of Slot Arbitration).

You can call `EMStatus` to determine whether the Event Manager is started. When it is, you can read keypresses by calling `GetNextEvent`. When the Event Manager is not started, you can read keys directly from the keyboard hardware by waiting for bit 7 of location $E0C000 to turn on. When it does, the lower seven bits represent the key pressed. Once you’ve detected a keypress, you need to write to location $E0C010 to remove the keypress from the buffer.

Alternately, you can use `IntSource` (in the Miscellaneous Tools) to temporarily disable keyboard interrupts and then read the keyboard hardware directly. Be sure to reactivate keyboard interrupts if, and only if, they were previously enabled.

### Just One Page of Stack Space

CDAs normally have only a single page of stack space available to them (256 bytes at $00/01xx). Your CDA may or may **not** be able to allocate additional stack space from bank 0 during execution. The following code (written for the MPW IIGS cross-assembler) shows a safe way to try to allocate more stack space and to switch between stacks when the space is available.

If ProDOS 8 is active, your CDA **cannot** allocate additional space (and there is no completely safe way to “borrow” bank 0 space from the ProDOS 8 application).

```
HowMuchStack   gequ    $1000          ;try for 4K of stack space

start          phd
               phb
               phk
               plb
               pha                    ;Space for result
               pha
               PushLong #HowMuchStack
               pha
               _MMStartUp
               pla
               ora    #$0f00          ;OR in an arbitrary auxiliary ID
               pha
               PushWord #$C001        ;fixed, locked, use specified bank
               PushLong #0            ;(specify bank 0)
               _NewHandle
               tsc
               sta    theOldStack
               bcs    NoStackSpace    ;still set from _NewHandle
               tcd
               lda    [1]
               tcd
;               clc                    ;carry is already clear
               adc    #HowMuchStack-1
NoStackSpace   pha
               ldx    #$fe
keepStack      lda    >$000100,x
               sta    stackImage,x
               dex
               dex
               bpl    keepStack
               pla
               tcs
               jsl    RealCDAentry     ;carry is clear if large stack available
               php
               php
               pla
               sta    pRegister
               sei
               ldx    #$fe
restoreStack   lda    stackImage,x
               sta    >$000100,x
               dex
               dex
               bpl    restoreStack
               lda    theOldStack
               tcs
               lda    pRegister
               pha
               plp
               plp
               lda    1,s
               ora    3,s
               beq    noDispose
               _DisposeHandle
               bra    Exit
noDispose      pla
               pla
Exit           plb
               pld
               rtl
pRegister     ds 2
theOldStack   ds 2
stackImage    ds.b 256
```

When this routine calls `RealCDAentry`, the carry flag is set if no extra stack space is available. If the carry is clear, the additional stack space was available and the direct-page register points to the bottom of that space.

```
RealCDAentry    bcs     smallStack              ;if c set, only 1 page of stack is
available
               ...                          ; put something interesting here
               rtl

smallStack     _SysBeep
               rtl
```

Note that interrupts are disabled while the page-one stack is being restored; they are reenabled (if they were originally enabled) only **after** the stack pointer is safely back in page one.

## Interrupts, Event Manager, Memory, and CDAs

Whether the Event Manager is active or not, the user hits Apple-Ctrl-Esc and usually gets to the CDA menu. It looks the same, but what happens internally is different affects what happens when your CDA allocates memory.

When the Event Manager is active (as it normally is while the user is running a Desktop application), hitting Apple-Ctrl-Esc posts a `deskAcc` event to the event queue. The CDA menu appears only when the application calls `GetNextEvent` or `EventAvail` with the `deskAcc` bit enabled in the event mask.

So with the Event Manager active, the CDA menu and individual CDAs are running in the "foreground"—no processor interrupt is being serviced, and the foreground application is stuck inside the `GetNextEvent` or `EventAvail` call. The Memory Manager knows that no interrupt is in progress, so it will happily compact and purge memory if necessary to carry out a memory allocation request from your CDA. This is just fine, since the foreground application made a toolbox call—unlocked memory blocks are not guaranteed to stay put.

When the Event Manager is not active, hitting Apple-Ctrl-Esc either enters the CDA menu immediately (if the system Busy Flag is zero) or calls `SchAddTask` so that the CDA menu appears during a the next `DECBUSYFLG` call that brings the system Busy Flag down to zero. If the CDA menu appears during a `DECBUSYFLG` , normal memory compaction and purging are possible, just like when the Event Manager is active.

But if the Busy Flag was zero when the user hit Apple-Ctrl-Esc, then the CDA menu appears inside of the interrupt, and the foreground application is at an unknown point where it may justifiably expect that unlocked memory blocks will not move or be purged (see *Apple IIGS Toolbox Reference*, Volume 1, page 12-5). (Note that the Desk Manager does a tricky dance to allow additional interrupts to occur, even though the Apple-Ctrl-Esc interrupt will not return until the user chooses Quit from the CDA menu. Normally interrupts cannot be nested; the Desk Manager and AppleTalk are exceptions.)

The Memory Manager knows an interrupt is in progress, so `CompactMem` takes no action and memory allocation requests do not cause unlocked memory blocks to move and do not attempt to purge purgeable blocks to make room. Memory allocation requests will still normally succeed, but you will not be able to allocate a block larger than the value returned by `MaxBlock`.

## New Desk Accessory Tips and Techniques

### An NDA Can Find its Menu Item ID

After the application has called `FixAppleMenu`, an NDA can look at its menu item template (after the "`\H`" in the NDA header) to determine the menu ID corresponding to the NDA's name in the Apple menu. This is sometimes useful to pass to `OpenNDA` (if the NDA has some way to open itself), or to pass to a Menu Manager call.

Finding the menu item ID in the NDA's header is easy if the NDA is written in assembly. In a high-level language it may be harder (if you don't have direct access to your NDA's header, you need to find it on the fly and scan for the "`\H`").

### NDAs and Command- Keystrokes

To give the user a consistent way to close NDA windows, System 6.0 handles Command-W automatically when a system window is in front. It calls `CloseNDAbyWinPtr` without letting the NDA or the application see the Command-W.

However, there is a special action code (`optionalCloseAction`) that an NDA can accept to handle the Close request itself. This way the NDA can offer the user a chance to cancel the Close, which is impossible when the system calls the NDA's main Close routine, as `CloseNDAByWinPtr` does. (See the System 6.0 Toolbox documentation for details.)

There is no way for an NDA to accept **some** keystrokes and pass others along to applications, but if your NDA does not want any keystroke events, turn off the corresponding `eventMask` bits in the NDA header (this allows the application to receive keystrokes while your NDA window is in front).

### Calling InstallNDA From Within an NDA

It is possible to write an NDA that installs other NDAs. However, with System Software 5.0 and later, `InstallNDA` returns an error when called from an NDA. When your NDA has control because the Desk Manager called one of your NDA's entry points, the Desk Manager's data structures are already in use, so `InstallNDA` is unable to modify them.

The solution is to use `SchAddTask` in the Scheduler to postpone the `InstallNDA` call until the system is not busy. Remember that the Bank and Direct Page registers are not defined when your scheduled task is executed.

### Processing mouseUp Events

When an NDA's action routine receives a `mouseUp` event, it is not always safe for the NDA to draw in its window.

For example, when the user drags an NDA window, the NDA receives the `mouseUp` before the window is actually moved, and before `DragWindow` erases the outline of the new window position, which may overlap the window's content. In addition, when the user chooses a menu item, the front NDA receives the `mouseUp` before the menu's image is removed, and the image may overlap the NDA's window. In either case, drawing in the window makes a mess.

The solution is to avoid drawing in direct response to a `mouseUp`. Instead, invalidate part of the window to force an update event to happen later.

## NDAs Can Have Resource Forks

Following is the recommended way for a New Desk Accessory to use its file's resource fork.

In the NDA's Open routine, do the following. Steps that are obsolete (and safely omitted) with System Software 6.0 and later are marked with an asterisk (*):

1. Call `GetCurResourceApp` and keep the result.
2. If the NDA does not already know its Memory Manager user ID, call `MMStartUp` to get it.
3. Call `ResourceStartUp` using the NDA's user ID.
4. Call the Loader function `LGetPathname2` with the NDA's user ID (and a `fileNumber` of $0001) to get a pointer to the NDA's pathname. (The result is a pointer to a class-one GS/OS string.)

*5. Use `GetLevel` to get the current file level, then use `SetLevel` to set it to zero. This helps protect your resource fork from being closed accidentally.

6. Use `GetSysPrefs` to get the current OS preferences, then use `SetSysPrefs` to ensure that the user is prompted, if necessary, to insert the disk containing your resource fork. (To compute the new preferences word, take the current one, `AND` it with $1FFF, and `ORA` it with $8000. This tells GS/OS to deal with volume-not-found conditions by putting up a please-insert-disk dialog with an OK button and a Cancel button.)
7. Call `OpenResourceFile` using the result from `LGetPathname2`. Save the returned `fileID`—you need it when closing the file. (Be prepared to deal with an error, such as $0045, Volume Not Found.)
8. Use `SetSysPrefs` to restore the OS preferences saved in step six.

*9. Use `SetLevel` to restore the file level to its old value (saved in step five).

10. Call `SetCurResourceApp` with the old value saved in step one.

In the NDA's action routine, no special calls are necessary—the Desk Manager calls `SetCurResourceApp` automatically before calling your action routine, so your NDA's own resource search path is already in effect.

Run queue routines and NDA installs with `AddToRunQ` are treated the same way—the NDA's resource search path is automatically in effect when the run queue routine is called.

In the NDA's Close routine, do the following:

1. Call `CloseResourceFile` with the `fileID` that was returned when you opened it.
2. Call `ResourceShutDown` with no parameters.

## NDAs Must Be Careful Handling Modal Windows

If your NDA uses its resource fork and calls `TaskMaster` with a restricted `wmTaskMask` to produce a modal window, you must be careful **not** to allow `TaskMaster` to update the contents of any application windows that happen to need updating.

The problem is that an application window's `wContDraw` routine can reasonably assume that the current Resource Manager search path is the application's, but `TaskMaster` does not take any special steps to set it. When the content-draw routine draws controls which were created from resources which are not presently in the resource search path, the system may crash.

If your NDA does not start up the Resource Manager, the Desk Manager is unable to `SetCurResourceApp` to your NDA, so the application's search path is still in effect—no problem. But if your NDA **does** start the Resource Manager, you have to be careful not to cause application routines to be called.

## Avoid Hard-Coding Your Pathname

If your NDA needs to know its own pathname or the pathname of the directory it's in, call `LGetPathname` or `LGetPathname2` using your User ID. This is a better method than hard-coding "*:System:Desk.Accs:MyDAName" because the user may change your DA's file name or use a utility to install it from some non-standard directory.

## Avoid Extra GetNewID calls

Normally there is no reason for a Desk Accessory to call `GetNewID`. When you can, just call `MMStartUp` to find your own User ID, and use that. You can freely use all the auxiliary IDs derived from your main ID (`MMStartUp`+$0100, `MMStartUp`+$0200, ..., `MMStartUp`+$0F00).

By not calling `GetNewID`, you conserve the limited supply of IDs (255 of in the $50xx range for Desk Accessories), and you make life easier for people trying to debug their systems, since all your allocated memory can be readily identified.

## Open is Not Called if NDA is Already Open

Your NDA's Open routine does not get called if the user chooses the NDA from the Apple menu while the NDA is already open. In this case, the Desk Manager simply calls `SelectWindow` on your existing window.

There is no need to include code in your Open routine to check if your window is already open, and to call `SelectWindow` if it is.

## Further Reference

- *Apple IIGS Toolbox Reference,* Volumes 1-3
- *GS/OS Reference*

- *Apple IIGS Hardware Reference*
- Apple IIGS Technical Note #53, Desk Accessories and Tools
- Apple IIGS Technical Note #57, The Memory Manager and Interrupts
- Apple IIGS Technical Note #69, The Ins and Outs of Slot Arbitration